Further Java Supo 1 - 1.8* (O)

A serialisation bomb, or rather a deserialisation bomb, works as follows:

When a DoS attack is mounted by causing the deserialisation of a short stream that requires a long time to deserialise. The stream is the deserialisation bomb.

The Problem - explained with example

```java
// Deserialisation bomb - deserialising this stream tasks forever

static byte[] bomb() {
    Set<Object> root = new HashSet<>();
    Set<Object> t1 = root;
    Set<Object> t2 = new HashSet<>();

    for Object i = 0 ; i < 100; i++) {
        Set<Object> t1 = new HashSet<>();
        Set<Object> t2 = new HashSet<>();
        t1.add("foo"); // make t1 != t2
        s1.add(t1);
        s1.add(t2);
        s2.add(t1);
        s2.add(t2);
        s1 = t1;
        s2 = t2;
    }
    return serialize(root); // Method omitted for brevity
}
```

The object graph contains 201 `HashSet` instances, each of which has 3 or fewer object references. It would take forever to deserialize this stream, as deserializing a `HashSet` instance requires computing the hash codes of its elements. The `hashCode` method is invoked over 2^{100} times.

Furthermore, the deserializer has no indication that anything is amiss. The stack eventually overflows.

The Solution

tl;dr use JSON, not Java Serialization

You open yourself up to attack whenever you deserialize byte streams you don't trust, therefore don't ever deserialize everything.

There is no reason to use Java serialization in any new system you write, because there are other mechanisms for translating between objects and byte sequences that avoid the dangers of Java serialization while offering advantages such as cross-platform support, high performance, huge ecosystem of tools and a broad community of expertise. These are ***cross-platform structured-data representations***, e.g. JSON and Protocol Buffers (Protobuf).

But I'm working with a legacy system!

So you can't avoid Java serialization. In this case, never deserialize untrusted data from untrusted sources.

But I don't trust my sources

Use object deserialization filtering (`java.io.ObjectInputFilter`). This allows you to specify a filter that is applied to data streams before they are deserialized. You can accept or reject certain classes via blacklisting or whitelisting. Whitelisting is better. A tool called *Serial Whitelist Application Trainer (SWAT)* can be used to automatically prepare a whitelist for your application. The filtering facility will also protect you against excessive memory usage and excessively deep object graphs but will not protect you against serialisation bombs like the one above.